Multi-GPU Training of ConvNets

Omry Yadan   **Keith Adams**   **Yaniv Taigman**   **Marc'Aurelio Ranzato**

Facebook AI Group
{omry, kma, yaniv, ranzato}@fb.com

Convolutional neural networks [3] have proven useful in many domains, including computer vision [1, 4, 5], audio processing [6, 7] and natural language processing [8]. These powerful models come at great cost in training time, however. Currently, long training periods make experimentation difficult and time consuming.

In this work, we consider a standard architecture [1] trained on the Imagenet dataset [2] for classification and investigate methods to speed convergence by parallelizing training across multiple GPUs. In this work, we used up to 4 NVIDIA TITAN GPUs with 6GB of RAM. While our experiments are performed on a single server, our GPUs have disjoint memory spaces, and just as in the distributed setting, communication overheads are an important consideration. Unlike previous work [9, 10, 11], we do not aim to improve the underlying optimization algorithm. Instead, we isolate the impact of parallelism, while using standard supervised back-propagation and *synchronous* mini-batch stochastic gradient descent.

We consider two basic approaches: data and model parallelism [9]. In data parallelism, the mini-batch is split across several GPUs as shown in fig. 3. Each GPU is responsible for computing gradients with respect to all model parameters, but does so using a subset of the samples in the mini-batch. This is the most straightforward parallelization method, but it requires considerable communication between GPUs, since each GPU must communicate both gradients and parameter values on every update step. Also, each GPU must use a large number of samples to effectively utilize the highly parallel device; thus, the mini-batch size effectively gets multiplied by the number of GPUs, hampering convergence. In our implementation, we find a speed-up of 1.5 times moving from 1 to 2 GPUs in the data parallel framework (when using 2 GPUs, each gets assigned a mini-batch of size 128). This experiment used the same architecture, network set up and dataset described in Krizhevsky et al. [1].

Model parallelism, on the other hand, consists of splitting an individual network's computation across multiple GPUs [9, 12]. An example is shown in fig. 4. For instance, a convolutional layer with N filters can be run on two GPUs, each of which convolves its input with N/2 filters. In their seminal work, Krizhevsky et al. [1] further customized the architecture of the network to better leverage model parallelism: the architecture consists of two "columns" each allocated on one GPU.

Table 1: Comparison of different parallelization schemes. All models use a mini-batch size equal to 256 samples. The network is a convolutional network with the same structure and hyper-parameter setting as described by Krizhevsky et al. [1] (with the only exception of the mini-batch size). The task is classification on the ImageNet 2012 datasset [2]. All GPUs are NVIDIA TITANs with 6GB of RAM and they all reside on the same server.

Configuration	Time to complete 100 epochs
1 GPU	10.5 days
2 GPUs Model parallelism	6.6 days
2 GPUs Data parallelism	7 days
4 GPUs Data parallelism	7.2 days
4 GPUs model + data parallelism	4.8 days

Columns have cross connections only at one intermediate layer and at the very top fully connected layers. While model parallelism is more difficult to implement, it has two potential advantages relative to data parallelism. First, it may requires less communication bandwidth when the cross connections involve small intermediate feature maps. Second, it allows the instantiation of models that are too big for a single GPUs memory. Our implementation of model parallelism, replicating prior work by Krizhevsky et al. [1], achieved a speed-up of 1.6 times moving from 1 to 2 GPUs.

Data and model parallelism can also be hybridized, as shown in fig. 5. We consider using 4 GPUs in three possible configurations: all model parallelism, all data parallelism, and a hybrid of model and data parallelism. In our experiment, we find that the hybrid approach yields the fastest convergence. The hybrid approach on 4 GPUs achieves a speed-up of 2.2 times compared to 1 GPU. More detailed results are shown in tab. 1. The convergence curves comparing the most interesting configurations are shown in fig. 1.

In general, not all configurations could be explored. For instance, on a single NVIDIA TITAN GPU with 6GB of RAM we are unable to fit mini-batches larger than 256 samples. On the other hand, we find mini-batches of 64 or fewer samples under-utilize the GPUs cores. This can be seen in tab. 1 which reports the timing of the data parallelism approach using 4 GPUs.

These preliminary results show promising speed-up factors by employing a hybrid parallelization strategy. In the future, we plan to extend this work to parallelization across servers by combining data and model parallelism with recent advances in asynchronous optimization methods and local learning algorithms.

Figure 1: Test error on the ImageNet dataset as a function of time using different forms of parallelism. On the y-axis we report the error rate of the test set, on the x-axis time in seconds. All experiments used the same mini-batch size (256) and ran for 100 epochs (here showing only the first 10 for clarity of visualization) with the same architecture and the same hyper-parameter setting as in Krizhevsky et al. [1]. If plotted against number of weight updates, all these curves would almost perfectly coincide.

References

[1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton. ImageNet classification with deep convolutional neural networks. In *NIPS*, 2012.

[2] J. Deng, W. Dong, R. Socher, L.J. Li, K. Li, and L. Fei-Fei. Imagenet: a large-scale hierarchical image database. In *CVPR*, 2009.

Figure 2: Diagram of a generic deep network. The number of arrows is proportional to the size of the mini-batch.

Figure 3: Diagram of a generic deep network using two GPUs (*data parallelism*). Each GPU computes errors and gradients for half of the samples in the mini-batch. Parameters and gradients are communicated across GPUs using PCI-e. The layers computed on a GPU all share the same color in the diagram.

[3] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[4] C. Szegedy, A. Toshev, and D. Erhan. Deep neural networks for object detection. In *NIPS*, 2013.

[5] C. Farabet, L. Couprie, C. amd Najman, and Y. LeCun. Learning hierarchical features for scene labeling. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2013.

[6] O. Abdel-Hamid, A.R. Mohamed, H. Jiang, and G. Penn. Applying convolutional neural networks concepts to hybrid nn-hmm model for speech recognition. In *ICASSP*, 2012.

[7] T. Sainath, A.R. Kingsbury, Mohamed, G. Dahl, G. Saon, H. Soltau, T. Beran, A. Aravkin, and B. Ramabhadran. Improvements to deep convolutional neural networks for lvcsr. In *ASRU*, 2013.

[8] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12:2493–2537, 2011.

[9] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, Q. Le, M. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang, and A. Ng. Large scale distributed deep networks. In *NIPS*, 2012.

[10] S. Zhang, C. Zhang, Z. You, R. Zheng, and B. Xu. Asynchronous stochastic gradient descent for dnn training. In *ICASSP*, 2013.

[11] X. Chen, A. Eversole, G. Li, D. Yu, and F. Seide. Pipelined back-propagation for context-dependent deep neural networks. In *Interspeech*, 2012.

[12] A. Coates, B. Huval, T. Wang, D.J. Wu, A.Y. Ng, and B. Catanzaro. Deep learning with cots hpc. In *ICML*, 2013.

Figure 4: Diagram of a generic deep network using two GPUs (*model parallelism*). The architecture is split into two columns which makes easier to split computation across the two GPUs.

Figure 5: Example of how model and data parallelism can be combined in order to make effective use of 4 GPUs.